MIQL: A Fortran Subroutine for Convex Mixed-Integer Quadratic Programming - User's Guide -

T. Lehmann, K. Schittkowski, T. Spickenreuther

Address: Department of Computer Science
University of Bayreuth
D - 95440 Bayreuth

Phone: +921 553278

E-mail: klaus.schittkowski@uni-bayreuth.de
Web: http://www.klaus-schittkowski.de
Date: May, 2010

Abstract

The Fortran subroutine MIQL solves strictly convex mixed-integer quadratic programming problems subject to linear equality and inequality constraints by a branch-and-cut method. At the root node of the branch-and-bound search tree, disjunctive and complemented mixed-integer rounding cuts are generated. The continuous subproblem solutions are obtained by the primal-dual method of Goldfarb and Idnani. The code is designed for solving small to medium size mixed-integer programs. Its usage is outlined and an illustrative example is presented.

Keywords: mixed-integer quadratic programming; quadratic optimization; MIQP; branch-and-bound; branch-and-cut; disjunctive cutting planes; complemented mixed-integer rounding cutting planes; numerical algorithms

1 Introduction

The Fortran subroutine MIQL solves strictly convex-mixed integer quadratic programming problems subject to linear equality and inequality constraints of the form

$$
\begin{aligned}
& \min \tfrac{1}{2}\left(x^T, y^T\right) C \begin{pmatrix} x \\ y \end{pmatrix} + d^T \begin{pmatrix} x \\ y \end{pmatrix} \\
& a_j^T \begin{pmatrix} x \\ y \end{pmatrix} + b_j = 0 \;, \qquad j = 1, \ldots, m_e \;\;, \\
x \in \mathbb{R}^{n_c}, y \in \mathbb{N}^{n_i} : \quad & a_j^T \begin{pmatrix} x \\ y \end{pmatrix} + b_j \geq 0 \;, \qquad j = m_e + 1, \ldots, m \;\;, \\
& x_l \leq x \leq x_u \;, \\
& y_l \leq y \leq y_u \;,
\end{aligned}
\tag{1}
$$

with an n by n positive definite matrix C, $n := n_i + n_c$, an n-dimensional vector d, an m by n matrix $A = (a_1, ..., a_m)^T$, and an m-vector b. Lower and upper bounds for the continuous and integer variables, x_l, x_u, y_l and y_u respectively, are separately handled. To ensure a consistant notation we introduce the two index sets I and J with $I \cup J = \{1, \ldots, n\}$ and $I \cap J = \emptyset$, where $y_k \in \mathbb{N}$ for $k \in I$, $|I| = n_i$, and $x_l \in \mathbb{R}$ for $l \in J$, $|J| = n_c$. The objective function of (1) is denoted by

$$
f(x) = \frac{1}{2}(x^T, y^T) C \begin{pmatrix} x \\ y \end{pmatrix} + d^T \begin{pmatrix} x \\ y \end{pmatrix} \;\; . \tag{2}
$$

If $n_i > 0$, the mixed-integer quadratic program (1) is solved by a branch-and-cut method. Branch-and-cut combines two solution strategies, a method for generating cutting planes and a branch-and-bound procedure. We apply cut generation only at the root node of the branch-and-bound search tree, i.e., before starting the enumeration process. For the present program version, we have implemented two different cut generation procedures known as complemented mixed-integer rounding cutting planes and disjunctive cutting planes.

This manual is organized as follows. The subsequent section introduces the basic concept of a branch-and-bound method and describes the different options of the underlying Fortran subroutine. Section 3 deals with the generation of cutting planes, whereas Section 4 describes more details of the implementation, especially the warm start features. Section 5 summarizes some numerical results to compare various options of the code. Further implementation details and program documentation are found in Section 6, followed by an illustrative example in Section 7.

2 The Branch-and-Bound Procedure

A branch-and-bound algorithm starts at the solution of the relaxed problem. In our case this is the corresponding continuous problem where the integrality condition $y_k \in \mathbb{N},\ \forall k \in I$, is removed. See also Dakin [17] for an early paper on mixed-integer linear programming. An integer variable $y_k \in I$ is selected and two different continuous subproblems are generated. They are obtained by rounding the continuous value of $y_k,\ k \in I$, to get two separate subproblems, one with upper bound $\lfloor y_k \rfloor$, another one with lower bound $\lfloor y_k \rfloor + 1$. Each subproblem determines a node in a binary search tree, which is internally created step by step.

A particular advantage of branch-and-bound methods is, that certain subtrees can be eliminated or *fathomed* at an early stage, before trying all possible combinations by exploring the whole tree. There are three situations where fathoming is possible:

- A subproblem is infeasible: All further subproblems obtained by branching from this node would also be infeasible, i.e., the node can be eliminated.
- The subproblem has a feasible integer solution: The corresponding optimal objective function value, if less than the known upper bound, provides a better upper bound for the optimal solution. Further branching from this node is not required.
- The continuous solution of a subproblem has a minimal solution value greater than the actual upper bound: Further branching from the node would only increase function values, and there is no chance to find a better integer solution. The node is eliminated.

A branch-and-bound method represents a general framework, which is easily adapted to other classes of mathematical optimization problems, say linear or nonlinear programming, see for example Gupta and Ravindran [22]. Essential is convexity of the original problem or the possibility to compute global solutions at all nodes. The integral node with minimal objective value represents the solution of (1) or we get the information that no feasible mixed-integer solution exists. Note that although the relaxed version of (1) is assumed to be a strictly convex optimization problem, the solution of the mixed-integer problem may not be unique. There is also a possibility to exploit lower bounds within a branch-and-bound algorithm by investigation the dual of (1), see Leyffer and Fletcher [25] for details.

Thus, there remain two open questions leading to different options, by which a branch-and-bound algorithm can be tuned:

1. Selecting an integer variable with a non-integer value in the continuous solution of a given node for branching:
   (a) *maximal fractional branching*: We select an integer variable from the solution of the relaxed subproblem with largest distance from next integer value.

(b) *minimal fractional branching*: We select an integer variable from the solution of the relaxed subproblem with smallest distance from next integer value.

2. Searching the tree for a free node from where branching, i.e., the generation of two new subproblems, is started:

   (a) *best of two*: The optimal objective function values of the two child nodes are compared and the node with a lower value is chosen. If both are leafs, i.e., if the corresponding solution is integral, or if the corresponding problem is infeasible or if there is already a better integral solution, strategy *best of all* is started.

   (b) *best of all*: All free nodes are compared and a node with lowest objective function value is selected.

   (c) *depth first*: This search strategy selects a child node whenever possible. If the node is a leaf the *best of all* strategy is applied.

The *depth first* strategy has the advantage, that the quadratic problem of a child node can be solved faster than any other one, since the only difference is one additional box constraint. Moreover, the search tree usually is smaller, i.e., there are less unexplored nodes. The disadvantage is, that the child node is arbitrarily selected leading to a larger number of subproblems to be solved.

Until the first integer feasible solution is found, the upper bound is infinity. After selecting an integer variable for branching and an unexplored not to continue, the optimal solutions of the corresponding relaxed quadratic programs are computed. This process is iterated until all nodes are either explored or fathomed.

3 Cut Generation

3.1 Motivation of Mixed-Integer Cutting Planes

Cutting plane methods for mixed-integer linear programming have been studied since a famous paper of Gomory [20] in 1958. The purpose of a cutting plane is to cut off the solution of a continuous relaxation of the original problem while retaining all feasible integer solutions. The combination of a branch-and-bound enumeration procedure and a cutting plane method is called branch-and-cut. Most cutting plane techniques relay either on the fact that the current iteration point to be cut off is a feasible basis solution, i.e., a corner of the feasible polyhedral set, or on special structures of some constraints or the whole feasible region. The tremendous possible speed-up for MILP problems is estimated by Bixby [12], see Table 1.

In general, the solution of a relaxed quadratic program does not need to be a basis solution at all, nor is special information about the constraints available in most cases. This

technique	speedup factor
cutting planes	53.7
presolve	10.8
branching rules	2.9
heuristics	1.4
node presolve	1.3
probing on dives	1.1

Table 1: Speed-up by Cutting in Linear Programming

is the main reason why there exists only very few results on cutting planes for solving mixed-integer quadratic programming (MIQP) problems of the form (1). Bienstock [11] suggests a branch-and-cut method based on disjunctive cuts for special MIQP problems, arising in portfolio optimization, where the structure of one single constraint is exploited to construct cuts.

In general, a cutting plane method tries to tighten the feasible region of a mixed-integer program. Although they are applied to solve convex nonlinear programs, see, e.g., Westerlund [34], their main application area are problems whose feasible regions are described by a polyhedron. Cutting plane methods successively add linear inequalities to the constraint set. They are called cutting planes and have to be valid for all feasible integral points, i.e., for every integral point satisfying the constraints of (1). On the other hand, they have to cut off the relaxed feasible solution $(\bar{x}, \bar{y})$ of a continuous relaxation. Cutting plane restrictions are added to the originally given set of constraints.

In the sequel, we generate cuts only at the root node of the branch-and-bound search tree, which implies that they are valid for each subproblem within the enumeration process. One reason is that the construction of these cutting planes is quite expansive and pays out only if they are globally valid. Furthermore the generation of cutting planes within the search process requires an efficient cut management and selection procedure, which is a challenging task. In our implementation cuts are constructed in rounds. We solve the relaxed original problem, then construct the cut planes and solve the tightened problem again. This process is continued until no more cuts are found or the maximal number of rounds is reached.

At the moment, two different cut generation procedures are implemented, which will be described in detail in the remainder of this section. First, we will describe complemented mixed-integer rounding cutting planes, which go back to Marchand and Wolsey [26]. Subsequently, we introduce disjunctive programming and the resulting disjunctive cuts, both heavily influenced by Balas [3].

3.2 Complemented Mixed-Integer Rounding Cuts

Complemented mixed-integer rounding (CMIR) cuts have been developed by Marchand and Wolsey [26], see also Goncalves and Ladanyi [21] and Wolters [35]. To simplify the notation, the m constraint indices are denoted by $P := \{1, \ldots, m\}$. To generate valid cuts, one has to transform (1) such that all variables are positive.

The generation of CMIR cutting planes consists of three steps: aggregation, bound substitution and cut generation. Within the aggregation step, a selection of original constraints is aggregated and, if possible, some continuous variables are eliminated. During bound substitution, continuous variables are substituted by $\hat{x}_j := x_{u_j} - x_j$ or $\hat{x}_j = x_{l_j} + x_j$. The resulting constraint is relaxed to yield a mixed knapsack set X^{MK} defined by

$$X^{MK} = \left\{ (s, y) \in \mathbb{R}_+ \times \mathbb{N}^{n_i} : \sum_{i \in I} \alpha_i' y_i \leq \beta' + s;\ y_i \leq y_{u_i},\ \forall i \in I \right\} \ , \tag{3}$$

where

$$s := - \sum_{j \in J, \gamma_j' < 0} \gamma_j' \hat{x}_j \ .$$

The original feasible set, X, is defined by

$$X := \left\{ (x, y) \in \mathbb{R}_+^{\hat{n}_c} \times \mathbb{N}^{n_i} : \sum_{i \in I} a_i^l y_i + \sum_{j \in J} a_j^l x_j = b^l,\ \forall l \in P;\ y_i \leq y_{u_i},\ \forall i \in I \right\} \ . \tag{4}$$

The inequality constraints of the original problem (1) are transformed into equalities by introducing slack variables within the CMIR cut generation procedure. The cut generation phase tries to find violated complemented mixed-integer rounding inequalities

$$\sum_{i \in T} F_{f_\tau}(\frac{a_i}{\delta}) y_i + \sum_{i \in U} F_{f_\tau}(-\frac{a_i}{\delta})(y_{u_i} - y_i) \leq \lfloor \tau \rfloor + \frac{s}{\delta(1 - f_\tau)} \tag{5}$$

for X^{MK} and to improve them. T and U are a partition of I and $\delta \in \mathbb{Q}_+ \setminus \{0\}$. Furthermore, we define

$$\tau := \frac{\sum_{i \in U} a_i y_{u_i}}{\delta}$$

and

$$F_\sigma(d) := \lfloor d \rfloor + \frac{(f_d - \sigma)^+}{1 - \sigma}$$

which is called the MIR function with $0 \leq \sigma < 1$ and $d^+ := \max\{d, 0\}$.

Marchand and Wolsey [26] show that the complemented mixed-integer rounding inequality (5) is always valid for X^{MK}. We also observe that X^{MK} is a relaxation of the original mixed-integer feasible region X and that every cut valid for X^{MK} is therefore valid for X.

If any complemented mixed-integer rounding inequality (5) is violated by the relaxed solution of MIQP (1), then we have fund a cutting plane. By varying the partition T and U and by trying out different values for δ, the quality of CMIR cuts can be improved. Different aggregated constraints can lead to different cuts. The drawback is, that there is no guarantee to find violated cuts by this procedure. But computational experience shows that it is often possible to find deep violated cuts.

Wolters [35] reports that complemented mixed-integer rounding cuts are the most successful cutting planes of the MILP solver SCIP of Achterberg [1]. For the code MIQL, we try to construct as many violated cuts as possible. Thus, we allow a large number of different starting constraints for the aggregation process. Together with some further modifications, we are able to construct significantly more cuts compared to the original aggregation scheme. The number of rounds is controlled by the user. A large number of cutting rounds is only beneficial for MIQP problems which are difficult to solve.

3.3 Disjunctive Cuts

Disjunctive cuts are cutting planes that are derived from a disjunction of the feasible region of a linearly constrained mixed-integer program. A disjunction in general decomposes the feasible region of the original optimization problem into disjoint polyhedra. One natural disjunction in the context of mixed-integer programming is a single disjunction for an integer variable y_k, where $k \in I$ is a fractional component of the relaxed solution $(\bar{x}, \bar{y})$,

$$(y_k \leq \lfloor \bar{y}_k \rfloor) \quad \vee \quad (y_k \geq \lceil \bar{y}_k \rceil). \tag{6}$$

A disjunction gives rise to a disjunctive relaxation of the original problem. The feasible region of this disjunctive relaxation is the union of the disjoint polyhedra corresponding to the considered disjunction. If the feasible region of the continuous relaxation of the original mixed-integer problem is given by

$$P \;:=\; \left\{ A \begin{pmatrix} x \\ y \end{pmatrix} \geq b;\; \begin{pmatrix} x \\ y \end{pmatrix} \geq 0 \right\} \tag{7}$$

the disjunctive relaxation corresponding to the standard disjunction (6) is the union of the polyhedra

$$\begin{aligned} P^k_{lower} &:= \left\{ A \begin{pmatrix} x \\ y \end{pmatrix} \geq b,\; \begin{pmatrix} x \\ y \end{pmatrix} \geq 0,\; y_k < \lfloor \bar{y}_k \rfloor \right\} \\ P^k_{upper} &:= \left\{ A \begin{pmatrix} x \\ y \end{pmatrix} \geq b,\; \begin{pmatrix} x \\ y \end{pmatrix} \geq 0,\; y_k > \lceil \bar{y}_k \rceil \right\}. \end{aligned}$$

In general, any optimization program with a constraint set consisting of the union of two or more polyhedra is called a disjunctive program.

Essential for the theory behind disjunctive programs is the existence of a compact representation of the convex hull of the union of polyhedra in a higher dimensional space. It can be projected back to the original space. This projection can be used to generate cuts which are called disjunctive cutting planes, see Balas [2, 3, 4, 5], Balas, Ceria, and Cornuejols [6], Balas and Perregaard [8, 9, 10], Ceria and Pataki [15], Ceria and Soares [16] for details. In practice, this is achieved by solving the so called cut generating linear program ($CGLP_k$) corresponding to disjunction (6),

$$\begin{array}{ll} & \min \quad a^T \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} - \beta \\ & a - A^T u + u_0 e_k \geq 0 \\ (a,\beta) \in \mathbb{R}^n \times \mathbb{R}, & a - A^T v - v_0 e_k \geq 0 \\ (u,u_0) \in \mathbb{R}^m \times \mathbb{R}, & -\beta + b^T u - u_0 \lfloor \bar{y}_k \rfloor = 0 \\ (v,v_0) \in \mathbb{R}^m \times \mathbb{R} & -\beta + b^T v + v_0 \lceil \bar{y}_k \rceil = 0 \\ & u,\ u_0,\ v,\ v_0 \geq 0 \end{array} \tag{8}$$

together with a normalization constraint of the form

$$\sum_i^m u_i + u_0 + \sum_i^m v_i + v_0 = 1 \ . \tag{9}$$

The linear program (8) constructs the cutting plane $a^T \begin{pmatrix} x \\ y \end{pmatrix} \geq \beta$ that is violated most at the current relaxed solution $(\bar{x}, \bar{y})$. The constraints of problem (8) restrict the choice to those cutting planes that are valid for the union of the polyhedra P^k_{lower} and P^k_{upper}. Therefore, the optimization variables (a, β) correspond to the resulting cutting plane, whereas the variables (u, u_0, v, v_0) ensure the validity of this constructed cut.

The cutting plane obtained by solving $CGLP_k$ only regards the integer condition on variable y_k. Therefore, this cut can be strengthened by a procedure of Balas and Jeroslow [7], that also considers the integrality condition on other variables. The result of solving $CGLP_k$ is one single violated cut. There exists various suggestions how to generate more than just one cut out of a disjunctive relaxation corresponding to disjunction (6), see Perregaard [28] for details. Furthermore, it is possible to restrict the disjunctive relaxation to a subspace of the original one neglecting all variables that are fixed at a bound. This reduces the size of $CGLP_k$ significantly. The resulting cutting plane of the reduced cut generating linear program can be lifted to be valid in the full space, see Perregaard [28].

In our current implementation of MIQL, we replace the linear program (8) by a quadratic

one $(CGQP_k)$,

$$
\begin{array}{ll}
\min & a^T \begin{pmatrix} \bar{x} \\ \bar{y}^* \end{pmatrix} - \beta + \epsilon \left(a^T a + \beta^2 + u^T u + u_0^2 + v^T v + v_0^2 \right) \\
& a - A^T u + u_0 e_k \geq 0 \\
& a - A^T v - v_0 e_k \geq 0 \\
(a, \beta) \in \mathbb{R}^n \times \mathbb{R}, & -\beta + b^T u - u_0 \lfloor \bar{y}_k \rfloor = 0 \\
(u, u_0) \in \mathbb{R}^m \times \mathbb{R}, & \\
(v, v_0) \in \mathbb{R}^m \times \mathbb{R} & -\beta + b^T v + v_0 \lceil \bar{y}_k \rceil = 0 \\
& \sum_i^m u_i + u_0 + \sum_i^m v_i + v_0 = 1 \\
& u,\ u_0,\ v,\ v_0 \geq 0,
\end{array}
\tag{10}
$$

where $\epsilon > 0,\ \epsilon \in \mathbb{R}$ is a suitable small value. Program (10) is solved by the quadratic solver QL of Schittkowski [31]. Furthermore our implementation of disjunctive cuts in MIQL creates and solves one cut generating quadratic program for each integer variable that is fractional at the relaxed solution of the mixed-integer quadratic program (1). All generated cuts are strengthened, if possible, by taking into account the integrality condition of all other integer variables. At present, we only generate one single cut per disjunctive relaxation. We restrict each $CGQP_k$ to the subspace of those variables that are not fixed at their lower bound with value zero. Afterwards, we lift the generated disjunctive cut to the full space, which significantly reduces the effort for generating disjunctive cuts.

4 Algorithmic Features of MIQL

As outlined in the previous sections, the two main components of a branch-and-cut approach are the branch-and-bound enumeration method and the cut generation procedures. In addition, we have to take into account the algorithm by which the relaxed subproblems, i.e., continuous convex quadratic programs are solved. The primal dual method of Goldfarb and Idnani [19] is perfectly suited for this task. For a better integration into the branch-and-cut framework we have extended the Fortran code QL of Schittkowski [31]. This extended version is called QLX and its calling sequence is described in the next section.

When called from MIQL, QLX performs warm starts depending on the relationship between successive subproblems, e.g., in a branch-and-bound procedure:

1. The new node is a child of the actual one, i.e., the new quadratic program is identical to the previous one apart from one additional box constraint subject to an integer variable. Since the primal-dual method of Goldfarb/Idnani successively adds constraints to the active set, we need in most cases only one additional iteration to get a new optimal solution. Even if more iterations are needed, almost always we save some calculation time compared to solving the complete quadratic program.

2. The new node has the same parent node as the actual one, i.e., both problems differ only in one box constraint for an integer variable. The algorithm identifies the solution of a quadratic program by an linear independent active set. A problem is solved by adding constraints to and eliminating constraints from the active set until all constraints are satisfied, and each iterate is dual feasible. To enable a warm start in the present situation, one has to ensure dual feasibility first. This is achieved by saving the active constraints of the common parent node. They are compared to the ones of the child, i.e., the last solved quadratic program. All other active constraints are eliminated. Then a dual feasible point is obtained and a new iteration of the algorithm is started by searching for a violated constraint. Time savings are achieved since in most cases, only few active constraints have to be deleted and added afterwards.

3. The new node is the child of a node which has the same parent node as the actual one (*nephew*): The warm start procedure is similar to the previous one. The only difference is that active constraints in the two child nodes are compared. This option is particularly efficient in case of the *best of two* strategy.

It is possible, that warm starts lead to numerical instabilities based on round-off errors. To avoid this situation, it is possible to limit the number of successive warm starts. In case of a numerical error, the problem is automatically resolved. Warm starts are indicated by parameter START of QLX that can obtain the following values:

START	Warm start performed by QLX
0	Normal QL run, no warm start performed.
10	Dual warm start, where only bounds are changed.
11	Dual warm start, where additional constraints are included.
30	Warm start and early termination, only one QL iteration to be performed.
31	Warm start and early termination, more than one QL iteration to be performed as provided by the parameter STEPS.
40	Remove active constraints from the active set, where the number of these constraints is stored in the calling parameter NRDC and the indices of the constraints to be removed are specified in the array DELC.
41	Remove active constraints from the active set analogue to START = 40, but continue afterwards with START = 30.
42	Remove active constraints from the active set analogue to START = 40, but continue afterwards with START = 31.
43	Remove active constraints from the active set analogue to START = 40, but continue afterwards with START = 10.

Table 2: Warm Start Modes

The term dual warm start indicates a situation, where a known solution is dual feasible for the subsequent continuous QP. START = 10 is applicable to a child problem during a branching step, while START = 11 is suitable for integrating cutting planes. By now we do not handle the situation where a previous solution is primal feasible, but not optimal. This situation can be bypassed by first removing all active constraints preventing dual feasibility.

Furthermore QLX can handle the following special QP formulation in an efficient way.

$$\begin{array}{lll} \min & \frac{1}{2}(x^T, z^T)\hat{C}\begin{pmatrix} x \\ z \end{pmatrix} + d^T \begin{pmatrix} x \\ z \end{pmatrix} & \\ x \in \mathbb{R}^{n_p}, & a_j^T x + \hat{a}_j^T z + b_j = 0, & j = 1, \ldots, m_e \ , \\ z \in \mathbb{R}^{n-n_p}: & a_j^T x + \hat{a}_j^T z + b_j \geq 0, & j = m_e + 1, \ldots, m \ , \\ & x_l \leq x \leq x_u, & \\ & z_l \leq z \leq z_u & \end{array} \tag{11}$$

$\hat{C}$ is an $n \times n$ matrix possessing the following structure:

$$\hat{C} \ = \ \begin{pmatrix} C & 0 \\ 0 & CD \end{pmatrix}, \tag{12}$$

where C is a $n_p \times n_p$ matrix and CD is a $(n - n_p) \times (n - n_p)$ diagonal matrix. $n_p \leq n$ can be considered as reduced problem dimension. $\hat{a}_j$ is the vector $AD(j) \cdot e_j$, where e_j is the j-th unit vector, i.e. the constraint matrix $\hat{A}$ with rows $(a_j^T, \hat{a}_j^T)$ exhibits the subsequent structure:

$$\hat{A} \ = \ \begin{pmatrix} A & AD \end{pmatrix}, \tag{13}$$

where A is a $m \times n_p$ matrix and AD is a $m \times (n - n_p)$ diagonal matrix. The diagonal elements of the matrices CD and AD are stored in the input arrays CDIAG and ADIAG and n_p has to be given to QLX by input parameter NP. This problem formulation (11) arises, e.g., when a quadratic program is relaxed by the introduction of m additional slack variables to ensure feasibility or in special data mining problems related to support vector machines, see Schittkowski [32].

In addition to the extended version of QL, there are some further improvements of our new version of MIQL:

1. The new *depth first* search benefits from warm starts.

2. The storing procedure for nodes in the branch-and-bound process is improved. Since a node marked by a fathoming rule is not needed any longer, its memory is used again for storing a new node.

3. Error handling is improved. In case of false termination when solving a quadratic program, the whole process is not terminated as in the previous version of MIQL. Instead, the error is treated in the same way as an infeasible leaf.

4. Improved bounds according to Leyffer and Fletcher [25] are calculated, if nodes are selected by *best of all*, using warm start options START = 30 and START = 41.

5. The branching direction for *depth first* search is determined by evaluating the Lagrangian function at $\lfloor \hat{y}_j \rfloor$ and $\lceil \hat{y}_j \rceil$, where j is the selected branching variable and $(\hat{x}, \hat{y})$ the solution of the previous branch-and-bound QP.

6. Lagrangian multipliers can be calculated independently of the solution method by a modified least squares approach, which uses an extended QP formulation to guarantee feasibility. This calculation is executed, if storage for $(2n + m + 4)^2$ branch-and-bound nodes is provided. Otherwise the returned multipliers depend on the solution strategy, i.e. the node selection strategy, cut generation and the branching rule. Independent multipliers are especially useful, if MIQL is used as subproblem solver for the mixed-integer nonlinear solver MISQP of Exler and Schittkowski [18], since then Lagrangian multipliers are needed to calculate a BFGS update.

7. Primal heuristics are now used for finding a first integer feasible solution before the start of the enumeration process. A diving and shifting procedure is implemented, which often finds a integer feasible solution close to the optimal one, by successively fixing one integer variable. Afterwards, the values of some variables are shifted according to the corresponding value of the partial derivative of the Lagrangian function. Furthermore, we have implemented a cheap routine to fix all integer variables by rounding them up or down and trying to retain feasibility.

5 Numerical Results

5.1 MISQP Subproblems

All numerical results are obtained on a Intel E6600 Dual Core CPU with 2.4 GHz under Windows XP 64 using Intel Fortran Compiler version 9.1.

For testing the nonlinear mixed-integer code MISQP, a large number of test problems have been implemented by Exler and Schittkowski [18]. Since in each iteration of the SQP-based method, a mixed-integer quadratic programming problem must be solved by MIQL, a large number of test examples is available. A subset of 35 MIQP subproblems is selected with calculation times over 0.1 seconds. Table 3 shows some characteristic data of the corresponding nonlinear programs.

Most of those test examples are selected from the GAMS MINLP-Library, see Bussieck, Drud, and Meeraus [14]. Some are simple case studies from petroleum industry. Prob-

problem	continuous variables	integer variables	inequality constraints	equality constraints
MISQP57	2	4	0	1
MISQP60	0	24	35	0
MISQP62	0	35	44	0
MISQP63	0	48	53	0
MISQP66	16	19	1	17
MISQP83	0	20	20	0
MISQP115	9	27	12	9
MISQP116	9	27	12	9
MISQP117	9	27	12	9

Table 3: Problem Characteristics

best of all		*best of two*		*depth first*	
nodes	time (sec)	nodes	time (sec)	nodes	time (sec)
115,588	36.3	121,996	26.5	122,055	26.4

Table 4: Node Selection Strategy

lems MISQP115, MISQP116 and MISQP117 have multiple global optima and the numerical results might become incompatible. In the subsequent tables, the total number of branch-and-bound nodes and the total calculation times are reported. For comparing all other options, we always choose the fastest alternative.

Table 4 shows the influence of different node selection strategies. Although *best of all* leads to the smallest amount of continuous QP problems that have to be solved during the branch-and-bound process, calculation time is considerably higher than for the other two strategies because of very few warm starts. Strategies *best of two* and *depth first* guided by the Lagrangian function value exhibit nearly identical performance.

Next, we consider the influence of warm starts subject to the node selection strategies *best of two* and *depth first*, see Table 5. For *best of all* warm starts can be neglected. We observe that warm starts improve the performance significantly.

The branching direction is chosen by the value of the Lagrangian function in case of *depth*

best of two				*depth first*			
warm start		no warm start		warm start		no warm start	
nodes	time (sec)	nodes	time (sec)	nodes	time (sec)	nodes	time (sec)
121,996	26.5	122,516	33.8	122,055	26.4	121,739	33.6

Table 5: Performance Improvements due to Warm Starts

Lagrangian		no Lagrangian	
nodes	time (sec)	nodes	time (sec)
122,055	26.4	302,145	44.5

Table 6: Branching Direction by Lagrangian Function

improved bounds		no improved bounds	
nodes	time (sec)	nodes	time (sec)
115,588	36.3	99,717	40.3

Table 7: Calculating Improved Bounds Information

first to prevent blind diving. Significant reduction of branch-and-bound nodes is achieved, see Table 6.

Table 7 shows the effect of calculating improved bounds leading to faster fathoming. The improved bounds exploit dual information and are calculated by an additional QL iteration. Improved bounds, see Leyffer and Fletcher [25], are only calculated in case of the *best of all* strategy. The results of Table 7 show that although the number of branch-and-bound nodes is reduced by early fathoming, the calculation of the improved bound information slows down the overall performance.

5.2 Benchmark Test Cases of Mittelmann

Next, we investigate the performance of MIQL for some of the benchmark test examples of Mittelmann [27]. Since MIQL is a dense solver and cannot exploit sparsity patterns, some of the problems are not suitable. In some other situations, the continuous relaxation is not solvable without special adjustments. Thus, we only consider examples ibell3a and imas284, see Table 8 for some data.

problem	constraints	number of variables	number of integer variables	non-zero density of constraint matrix	non-zero density of objective function matrix
ibell3a	104	122	60	3.1%	0.4%
imas284	68	151	150	95.3%	0.6%

Table 8: Characteristics of Mittelmann Test Examples

Numerical results, i.e., number of nodes and calculation time, are shown in Table 9. Cuts are not generated. The option *depth first* is superior to the other node selection strategies, since it benefits from warm starts. Furthermore, an integer feasible solution is often found quickly, enabling fathoming of branch-and-bound branches.

problem	branching strategy	node selection	nodes	time (sec)
ibell3a	*maximal fraction*	*best of all*	61,588	1,201
ibell3a	*maximal fraction*	*best of two*	63,777	1,125
ibell3a	*maximal fraction*	*depth first*	74,720	952
imas284	*maximal fraction*	*best of all*	142,241	3,610
imas284	*maximal fraction*	*best of two*	186,412	3,576
imas284	*maximal fraction*	*depth first*	142,434	1,940

Table 9: Results for Mittelmann Test Examples without Cuts

Table 10 includes the number of cuts and the reduced gap. Calculation time is significantly reduced by about 60% for problem ibell3a and 23% for imas284.

problem	branching strategy	node selection	nodes	time (sec)	cuts	reduced gap
ibell3a	*maximal fraction*	*depth first*	29,808	381	16	51.5%
imas284	*maximal fraction*	*depth first*	127,849	1,494	58	11.6%

Table 10: Results for Mittelmann Test Examples with Cuts

6 Program Documentation

The Fortran subroutine MIQL solves the mixed-integer quadratic program (1) with a positive definite objective function matrix by calling the general purpose branch-and-bound code BFOUR, see Lehmann, Schittkowski, and Spickenreuther [24]. Different rules for selecting variables and branching strategies are available. Prior to the call of BFOUR, the cut generation routines are executed. The complemented mixed-integer rounding cuts are constructed by subroutine CMIR based is the corresponding cut generation routine of Achterberg [1]. They are adapted, for example, to construct more cuts than the original implementation. The routine CMIR is called in combination with the routine DJCUT, which generates cuts for each fractional variable by calling QLX, to solve the corresponding cut generation quadratic program (10). The necessary storage for generating the quadratic program is taken from the one that is reserved for the branch-and-bound process, since the cuts are generated prior to the enumeration process. The number of $(2n + 2m + 6)^2$ maximum branch-and-bound nodes guarantees the generation of disjunctive cuts. If sufficient storage is not available, the generation of disjunctive cuts is omitted. CMIR and disjunctive cutting plane generation are both executed within a given maximum number of rounds. A suitable value depends heavily on the complexity of the MIQP problem.

The relaxed continuous quadratic program (1) is solved by an extention of the code QL of Schittkowski [31], which is based on the primal-dual method of Goldfarb and Idnani [19], see

also Powell [29] for an efficient implementation or Boland [13] for an extension to the semi-definite case. Initially, a Cholesky decomposition of C is computed by an upper triangular matrix R such that $C = R^T R$. In case of numerical instabilities, e.g., round-off errors, or a semi-definite matrix C, a certain multiple of the unit matrix is added to C to get a positive definite matrix for which a Cholesky decomposition is computed. Subsequently, the known triangular factor is saved and used again, whenever solving another relaxed subproblem. Successively, violated constraints are added to an *active set* until a solution is obtained. In each step, the minimizer of the objective function subject to the new active set is computed. If an iterate satisfies all linear constraints and bounds, the optimal solution is obtained and the algorithm terminates.

A particular advantage of a dual method is that phase I of a primal algorithm, i.e., the computation of an initial feasible point satisfying all linear constraints and bounds, can be avoided. The algorithm proves its robustness and efficiency in the framework of sequential quadratic programming algorithm, see Schittkowski [30, 33]. A further advantage is the possibility of handling warm starts efficiently as pointed out in the previous section.

The calling sequence and the meaning of the parameter settings are described below, where default values, as far as applicable, are set in brackets.

Usage:

```
    CALL  MIQL (     M,     ME,   MMAX,      N,   NMAX,
   /                MNN,     NI,    IND,      C,      D,
   /                  A,      B,     XL,     XU,      X,
   /                  U,      F,    ACC,    EPS,   IOPT,
   /               ROPT,   LOPT,   IOUT,  IFAIL, IPRINT,
   /                 RW,    LRW,     IW,    LIW,     LW,
   /                LLW                                 )
```

Parameter Definition:

M : Input parameter for the total number of constraints including equality constraints.

ME : Input parameter for the number of equality constraints.

MMAX : Input parameter defining the row dimension of array A containing linear constraints, must be at least M + CMAX, CMAX maximum number of cuts.

N :	Input parameter for the number of optimization variables.
NMAX :	Input parameter defining the row dimension of C, at least one and greater or equal to N.
MNN :	Input parameter equal to MMAX+N+N when calling QL, dimension of U.
NI :	Input parameter for the number of integer variables.
IND(NI) :	Integer input vector for indices of integer variables.
C(NMAX,N) :	Double precision input matrix for objective function matrix.
D(N) :	Double precision input vector for constant vector of objective function.
A(MMAX,N) :	Double precision input and output matrix for linear constraints, first ME rows for equality, then M-ME rows for inequality constraints, last MMAX-M rows set to cut constraints on return.
B(MMAX) :	Double precision input and output vector for constant values of linear constraints and the cut constraints on return.
XL(N),XU(N) :	Double precision input vectors for upper and lower bounds of the variables.
X(N) :	Double precision output vector for the optimal solution.
U(MNN) :	Double precision output vector for the multipliers subject to linear constraints at the first M positions and for bounds, but without cut constraints.
F :	Double precision output parameter containing optimal objective function value.
ACC :	Double precision input parameter for identifying integer values, set to machine precision in case of ACC=0.
EPS :	Double precision input parameter for termination accuracy of the QP solver QL, should not be smaller than machine precision, set to machine precision in case of EPS=0.
IOPT(20) :	Integer option array:
IOPT(1)	Branching rule, 1 : Maximal fractional branching. 2 : Minimal fractional branching.
IOPT(2)	Node selection strategy, 1 : Best of all.

2 : Best of two.
3 : Depth first.

IOPT(3) Maximal number of nodes, should be at least (2*N+2*M+6)**2 to guarantee generation of disjunctive cuts, and must be at least (2N+M+4)**2 for recomputing Lagrange multipliers (100,000).

IOPT(4) Maximal number of successive warmstarts to avoid numerical instabilities (100).

IOPT(5) Calculate improved bounds if *best-of-all* selection strategy is used (0).

IOPT(6) Select direction for *depth-first* according to value of Lagrange function (0).

IOPT(7) Cholesky decomposition mode (1),

0 : Calculate Cholesky decomposition once and reuse it.

1 : Calculate new Cholesky decomposition whenever warmstart is not activate.

IOPT(8) Control the cutting process (0),

0 : No cuts.
1 : Disjunctive cuts only.
2 : Complemented mixed integer rounding (CMIR) cuts only.
3 : Both disjunctive and CMIR cuts.

IOPT(9) Maximal number of cycles for disjunctive cuts (1).

IOPT(10) Maximal number of cycles for CMIR cuts (1).

IOPT(11) Primal heuristic mode (0),

0 : No primal heuristics.
1 : Nearest integer.
2 : Feasibility pump.

IOPT(12) Input parameter for the reduced quadratic program dimension (N).

IOPT(13) Output parameter for total number of generated cuts.

IOPT(14) Output parameter for total number of branch-and-bound nodes.

ROPT(20) : Double precision option array:

ROPT(1) Output parameter for time spent for generating cuts.

ROPT(2) Output parameter for time spent for branch-and-bound.

ROPT(3) Output parameter for gap reduced by cutting planes compared to original relaxed solution.

ROPT(4) Output parameter for relaxed optimal objective value.

LOPT(20) : Logical option array:

LOPT(1) Output parameter indicating cut generation.

IOUT : Input parameter for desired output unit number, i.e., all write-statements start with 'WRITE(IOUT,... '.

IFAIL : Output parameter for termination reason:

-4 : Branch-and-bound root QP not solvable.

-3 : Relaxed QP without feasible solution.

-2 : Feasible solution not within maximal number of nodes.

-1 : Feasible integer solution does not exist.

0 : Optimal solution found.

1 : Feasible solution found, but the maximum number of nodes reached.

3 : Termination tolerance ACC negative.

4 : Length of a working array RW, IW, or LW too small.

5 : Parameters N, M, MNN, ME, or NI incorrect.

6 : Option of IOPT incorrectly set.

7 : Lagrangian multipliers could not be calculated.

8 : IOUT or IPRINT incorrectly set.

9 : Lower variable bound greater than upper one.

IPRINT : Input parameter defining the output level:

0 : No output to be generated.

1 : Output of relaxed solution and final performance summary.

2 : Intermediate reduced, i.e. one line, output for all generated subproblems, i.e., all nodes, including node status:

* : fractional feasible node,

B : new best integer feasible solution,

I : infeasible node,

M : marked node,

	3 : Complete output for all nodes, e.g., with bounds.
RW(LRW) :	Double precision working array of length LRW
LRW :	Input parameter for length of RW, must be at least 5*NMAX*NMAX/2 + 7*NI + 10*NMAX + 16*MMAX + 15*N + 3*MAXNDS + 14
IW(LIW) :	Integer working array of length LIW
LIW :	Input parameter for length of IW, must be at least 4*MMAX + NMAX + 5*NI + 6*MAXNDS + 6*N + 11
LW(LLW) :	Logical working array of length LLW
LLW :	Input parameter for length of LW, must be at least MMAX + N + 3*NI + 3

The continuous quadratic programming solver QL, see Schittkowski [31], is extended to satisfy the requirements of Section 4.

Usage:

```
      CALL  QLX (     M,      ME,    MMAX,       N,    NMAX,
     /                MNN,     C,     D,          A,    B,
     /                XL,      XU,    X,          U,    EPS,
     /                MODE,    IOUT,  IFAIL,  IPRINT,    RW
     /                LRW,     IW,    LIW,        NP,   ADIAG,
     /                CDIAG,   START, NACT,    DELC,     NRDC,
     /                STEPS                                )
```

Definition of the parameters:

M :	Number of constraints.
ME :	Number of equality constraints.
MMAX :	Row dimension of array A containing linear constraints. MMAX must be at least one and greater or equal to M.
N :	Number of optimization variables.
NMAX :	Row dimension of C. NMAX must be at least one and greater or equal to N.

MNN :	Must be equal to M+N+N when calling QL, dimension of U.
C(NMAX,N) :	Objective function matrix which should be symmetric and positive definite. If MODE=0, C is supposed to be the upper triangular factor of a Cholesky decomposition of the objective function matrix.
D(N) :	Contains the constant vector of the quadratic objective function.
A(MMAX,N) :	Matrix of the linear constraints, first ME rows for equality, then M-ME rows for inequality constraints.
B(M) :	Constant values of linear constraints in the same order.
XL(N),XU(N) :	On input, the one-dimensional arrays XL and XU must contain the upper and lower bounds of the variables.
X(N) :	On return, X contains the optimal solution.
U(MNN) :	On return, U contains the multipliers subject to the linear constraints (first M locations) and bounds. At the optimal solution, all multipliers of inequality constraints are nonnegative.
EPS :	Final termination accuracy (e.g. 1.0D-12). The parameter value should not be smaller than the underlying machine precision.
MODE :	0 : Initial Cholesky factorization of C provided by the calling program, stored in the upper triangular part of array C. 1 : Cholesky decomposition internally computed.
IOUT :	Output unit number, i.e., all write-statements start with 'WRITE(IOUT,... '.
IFAIL :	Termination reason: 0 : Optimality conditions satisfied. 1 : Termination after too many iterations (40*(N+M)). 2 : Termination accuracy insufficient. 3 : Inconsistency, division by zero. 5 : Length of a working array too short. >100 : Inconsistent constraints and IFAIL=100+ICON, where ICON denotes a constraint causing the conflict.
IPRINT :	Specification of the desired output level: 0 : No output of the program. 1 : Only a final error message is given.
RW(LRW) :	Double precision working array of length LRW

LRW :	Input parameter for length of RW, must be at least 3*NMAX*NMAX/2 + 10*NMAX + 2*MMAX + 1.
IW(LIW) :	Integer working array of length LIW
LIW :	Input parameter for length of IW, must be at least N
NP:	Input parameter defining the dimension of the reduced problem (11).
ADIAG:	Input array defining the diagonal entries of the constraint matrix of the reduced problem formulation (11).
CDIAG:	Input array defining the diagonal entries of the objective matrix of the reduced problem formulation (11).
START:	Input parameter specifying the type of warmstart to be performed, more details are given in Section 4.
NACT:	Input and output parameter indicating the number of active constraints. For a correct warm start it has to possess the value corresponding to the solution of the previous quadratic program.
DELC(NRDC):	Input array containing the indices of the active constraints to be removed by a corresponding warm start.
NRDC:	Input parameter defining the number of active constraints to be deleted by a corresponding warm start. Therefore it has to be smaller than or equal to NACT.
STEPS:	Input parameter defining the number of QL iterations to be performed before an early termination.

7 Example

To give an example, we consider the simple quadratic program

$$\begin{array}{ll} & \min \ \ \frac{1}{2}\sum_{i=1}^{5} x_i^2 - 21.98x_1 - 1.26x_2 + 61.39x_3 + 5.3x_4 + 101.3x_5 \\ x_1, x_2 \in \mathbb{R}, & \\ x_3, x_4, x_5 \in \mathbb{N}: & -7.56x_1 + 0.5x_5 + 39.1 \geq 0 \ , \\ & -100 \leq x_i \leq 100 \ , \quad i = 1 \ldots, 5 \end{array}$$

The corresponding Fortran source code is listed below.

```
      IMPLICIT NONE
      INTEGER          NMAX,MMAX,MNNMAX,LIWMAX,LWMAX,LLWMAX,MAXNDS
      PARAMETER       (NMAX   = 5,
     /                 MMAX   = 10,
     /                 MAXNDS = 1000,
     /                 MNNMAX = MMAX + NMAX + NMAX,
     /                 LIWMAX = 16*NMAX + 6*MAXNDS + 4*MMAX + 11,
     /                 LWMAX  = 5*NMAX*NMAX/2 + 32*NMAX + 16*MMAX
     /                          + 3*MAXNDS + 14,
```

```
     /                 LLWMAX = MMAX + 4*NMAX + 3)
      INTEGER          I,J,IPRINT,IOUT,N,M,ME,NI,MNN,IOPT(20),
     /                 IFAIL,IM(NMAX),IW(LIWMAX)
      DOUBLE PRECISION FMIN,ACC,EPS,A(MMAX,NMAX),C(NMAX,NMAX),B(MMAX),
     /                 D(NMAX),XL(NMAX),XU(NMAX),X(NMAX),U(MNNMAX),
     /                 W(LWMAX),ROPT(20)
      LOGICAL          LW(LLWMAX),LOPT(20)
C
C   Set parameters
C
      N         = 5
      M         = 1
      ME        = 0
      MNN       = MMAX + N + N
      NI        = N
      ACC       = 1.0D-10
      EPS       = 1.0D-15
      IOPT(1) = 1
      IOPT(2) = 3
      IOPT(3) = MAXNDS
      IOPT(4) = 1000
      IOPT(5) = 1
      IOPT(6) = 1
      IOPT(7) = 1
      IOPT(8) = 3
      IOPT(9) = 1
      IOPT(10)= 1
      IOPT(11)= 2
      IOPT(12)= N
      IPRINT    = 2
      IOUT      = 6
C
C   Set problem data
C
      DO I=1,N
         DO J=1,N
            C(I,J) = 0.0D0
         ENDDO
         C(I,I) =    1.0D0
         A(1,I) =    0.0D0
         XL(I)  = -100.0D0
         XU(I)  =  100.0D0
      ENDDO
      DO I=1,NI
         IM(I) = I
      ENDDO
      A(1,1) = -7.56D0
      A(1,5) =  0.50D0
      B(1)   =  39.1D0
      D(1)   =-21.98D0
      D(2)   = -1.26D0
      D(3)   = 61.39D0
      D(4)   =   5.3D0
      D(5)   = 101.3D0
C
C   Calling MIQL
C

      CALL MIQL(M,ME,MMAX,N,NMAX,MNN,NI,IM,C,D,A,B,XL,XU,X,U,
     /          FMIN,ACC,EPS,IOPT,ROPT,LOPT,IOUT,IFAIL,IPRINT,
     /          W,LWMAX,IW,LIWMAX,LW,LLWMAX)
C
```

```
      STOP
      END
C
```

The following output should appear on screen:

```
   MIQL - A branch-and-cut mixed-integer quadratic solver

  *** INFO (MIQL): Improved bounds calculation deactivated!

     Parameters:
        Maximal branch-and-bound nodes    =         1000
        Node selection strategy           =            3
        Branching variable selection      =            1
        Improved bounds calculation       =            0
        Lagrangian direction              =            1
        QL decomposition mode             =            1
        Cut generation mode               =            3
        Primal heuristic mode             =            2
        Maximal successive warmstarts     =         1000
        Maximal number of cutting planes  =            9
        Disjunctive cutting rounds        =            1
        CMIR cutting rounds               =            1

   Problem transformed to positive orthant!


   BFOUR - A branch and bound framework

     Parameters:
        Number of integer variables       =            5
        Maximal number of nodes           =         1000
        Integer tolerance                 = 0.100000D-09
        Branching variable selection      =            1
        Node selection strategy           =            3
        Convex problem
        Initial integer feasible solution =  -0.1750D+05

     Output in the following order:
        S     - status of current node
        IT    - iteration number
        ND    - index of current node
        F     - objective function value
        P     - index of parent node
        LB    - lower bound
        UB    - upper bound

   S      IT       ND         F           FA        LB           UB
 ------------------------------------------------------------------------
   *        1        1  -.175108D+05        0  -.175108D+05  -.175035D+05
   *        2        2  -.175107D+05        1  -.175108D+05  -.175035D+05
   *        3        3  -.175082D+05        2  -.175108D+05  -.175035D+05
   *        4        4  -.175081D+05        3  -.175108D+05  -.175035D+05
   B        5        5  -.175081D+05        4  -.175108D+05  -.175081D+05
   *        6        5  -.175106D+05        1  -.175108D+05  -.175081D+05
   M        7        6  -.175081D+05        5  -.175108D+05  -.175081D+05
   M        8        6  -.175007D+05        2  -.175107D+05  -.175081D+05
   M        9        6  -.175005D+05        5  -.175106D+05  -.175081D+05
```

```
  M       10        6  -.175079D+05        3  -.175082D+05  -.175081D+05
  M       11        6  -.175079D+05        4  -.175081D+05  -.175081D+05


------------------------------------------------------------------------
  Optimal solution:
------------------------------------------------------------------------
  Number of explored subproblems:       11
  Highest used index:         5
  F(Y)   =   -17508.0900000
    Y(   1) =    98.000000
    Y(   2) =    101.00000
    Y(   3) =    39.000000
    Y(   4) =    95.000000
    Y(   5) =    0.0000000

       QPs exposed to numerical problems =            0
       Generated cutting planes          =            2
       Reduced gap                       = 0.797323D+00

  Optimal solution of original problem:

  Objective value   =   -6983.09000000
    X(   1) =   -2.0000000
    X(   2) =    1.0000000
    X(   3) =   -61.000000
    X(   4) =   -5.0000000
    X(   5) =   -100.00000
```

Some further quadratic programs for testing a correct implementation are found in Hock and Schittkowski [23].

References

[1] Achterberg T. (2004): *SCIP - a framework to integrate constraint and mixed-integer programming*, Technical Report 04-19, Zuse Institute Berlin, Germany

[2] Balas E. (1971): *Intersection cuts - a new type of cutting planes for integer programming*, Operations Research, Vol. 19, 19-39

[3] Balas E. (1975): *Disjunctive programming - cutting planes from logical conditions*, Nonlinear Programming, Vol. 2, 330-382

[4] Balas E. (1979): *Disjunctive programming*, Annals of Discrete Mathematics, Vol. 5, 3-51

[5] Balas E. (1998): *Disjunctive programming: Properties of the convex hull of feasible points*, Discrete Applied Mathematics, Vol. 89, 1-44

[6] Balas E., Ceria S., Cornuejols G. (1993): *A lift-and-project cutting plane algorithm for mixed 0-1 programs*, Mathematical Programming, Vol. 58, 295-324

[7] Balas E., Jeroslow R. (1980): *Strengthening cuts for mixed-integer programs*, European Journal of Operations Research, Vol. 4, 224-234

[8] Balas E., Perregaard M. (2001): *Generating cuts from multiple-term disjunctions*, Lecture Notes in Computer Science, Vol. 2081, 348-360

[9] Balas E., Perregaard M. (2002): *Lift-and-project for mixed 0-1 programming: recent progress*, Discrete Applied Mathematics, Vol. 123, 129-154

[10] Balas E., Perregaard M. (2003): *A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed-integer gomory cuts*, Mathematical Programming, Vol. 94, 221-2245

[11] Bienstock D. (1994): *Computational study of a family of mixed-integer quadratic programming problems*, Columbia University New York, USA

[12] Bixby R.E., Fenelon M., Gu Z., Rothberg E., and Wunderling R. (2004): *Mixed-integer programming: A progress report*, in Martin Grötschel (ed.), *The sharpest cut: The impact of Manfred Padberg and his work*, MPS-SIAM Series on Optimization, Vol. 4

[13] Boland N.L. (1997): *A dual-active-set algorithm for positive semi-definite quadratic programming*, Mathematical Programming, Vol. 78, 1-27

[14] Bussieck M.R., Drud A.S., Meeraus A. (2007): *MINLPLib - A collection of test models for mixed-integer nonlinear programming*, GAMS Development Corp., Washington D.C., USA

[15] Ceria S., Pataki G. (1998): *Solving integer and disjunctive programs by lift-and-project*, Lecutre Notes in Computer Science, Vol. 1412, 271-283

[16] Ceria S., Soares J. (1997): *Disjunctive cut generation for mixed 0-1 programs: duality and lifting*, GSB, Columbia University New York, USA

[17] Dakin R.J. (1965): *A tree search algorithm for mixed-integer programming problems*, Computer Journal, Vol. 8, 250-255

[18] Exler O., Schittkowski K. (2006): *MISQP: A Fortran implementation of a trust region SQP algorithm for mixed-integer nonlinear programming - User's guide, version 2.0*, Report, Department of Computer Science, University of Bayreuth, Germany

[19] Goldfarb D., Idnani A. (1983): *A numerically stable method for solving strictly convex quadratic programs*, Mathematical Programming, Vol. 27, 1-33

[20] Gomory, R.E. (1958): *An algorithm for integer solutions to linear programms*, Princeton I.B.M. Mathematics Research Project, Technical Report No. 1, Princeton University, USA

[21] Goncalves J.P.M., Ladanyi L. (2005): *An implementation of a separation procedure for mixed-integer rounding inequalities*, IBM Research Report RC23686 (W0508-022)

[22] Gupta O.K., Ravindran A. (1985): *Branch-and-bound experiments in convex nonlinear integer programming*, Management Science, Vol. 31, 1533-1546

[23] Hock W., Schittkowski K. (1981): *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems, Vol. 187, Springer

[24] Lehmann T., Schittkowski K., Spickenreuther T. (2008): *BFOUR: A Fortran subroutine for integer optimization by branch-and-bound - user's guide* -, Report, Department of Computer Science, University of Bayreuth, Germany

[25] Leyffer S., Fletcher R. (1998): *Numerical experiments with lower bounds for MIQP branch-and-bound*, SIAM Journal on Optimization, Vol. 8, 604-616

[26] Marchand H., Wolsey L.A. (2001): *Aggregation and mixed-integer rounding to solve mixed-integer programs*, Operations Research, Vol. 49, 363-371

[27] Mittelmann H.(2007): *Mixed-integer (QC) QP benchmark*, http://plato.asu.edu/ftp/miqp.html

[28] Perregaard M. (2003): *Generating disjunctive cuts for mixed-integer programs*, Doctoral Dissertation, Carnegie Mellon University Pittsburgh, USA

[29] Powell M.J.D. (1983): *ZQPCVX, A Fortran subroutine for convex quadratic programming*, Report DAMTP/1983/NA17, University of Cambridge, England

[30] Schittkowski K. (1985/86): *NLPQL: A Fortran subroutine solving constrained nonlinear programming problems,* Annals of Operations Research, Vol. 5, 485-500

[31] Schittkowski K. (2003): *QL : A Fortran code for convex quadratic programming - User's guide*, Report, Department of Mathematics, University of Bayreuth, Germany

[32] Schittkowski K. (2005): *Optimal parameter selection in support vector machines*, Journal of Industrial and Management Optimization, Vol. 1, 465-476

[33] Schittkowski K. (2006): *NLPQLP: A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search - user's guide, version 2.2*, Report, Department of Computer Science, University of Bayreuth, Germany

[34] Westerlund T., Petterson F. (1995): *An extended cutting plane mathod for solving convex MINLP problems*, Computers and Chemical Engineering, Vol. 21, 131-136

[35] Wolters K. (2006): *Implementation of cutting plane separators for mixed-integer programs*, Diploma Thesis, Zuse Institute Berlin, Germany